5-25-01

AF #175 Bond 6/1/01 2151

Express Mail No.: EL 501 639 990 US

APPEAL TO THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application of: Burke

Application No.: 09/025,143

Group Art Unit: 2755

Filed: February 18, 1998

Examiner: Lao, S.

For: Foreign Object Definition Information Repository

Attorney Docket No.: 9318-003-999

APPELLANT'S AMENDED BRIEF ON APPEAL

Garland T. Stephens, Esq.
PENNIE & EDMONDS LLP
1155 Avenue of the Americas
New York, New York 10036-2711
(212) 790-9090

TABLE OF CONTENTS

This is an appeal from the decision dated February 4, 2000 of the Examiner finally rejecting Claims 1-21 of U.S. Application of Burke, Serial No. 09/025,143 ("the Burke Application"). Appellant's Notice of Appeal was timely filed on August 4, 2000, with appropriate extensions of time.

This Amended Brief is filed in response a Patent Office communication mailed April 24, 2001 stating that Appellant's timely filed appeal brief did not comply with the requirements of 37 C.F.R. § 1.192 because it did not provide grouping for every claim. As set forth below, this Amended Brief clarifies Appellant's grouping of claims and Appellant's contention that the claims do not all stand or fall together. This Amended Brief is timely filed within the time period set by the Patent Office communication.

I. Real Party In Interest

Iona Technologies, PLC is the assignee of the Burke application. Iona Technologies, PLC and its United States affiliate, Iona Technologies, Inc. (collectively, "Iona") are the real parties in interest. An assignment from Mark Burke to Iona Technologies, PLC has been recorded by the Assignment Division of the United States Patent Office at Reel/Frame 10129/0899.

II. Related Appeals and Interferences

There are no appeals or interferences known to Appellant, its legal representative or assignee which will affect or be directly affected by, or have a bearing on the Board's decision in this pending appeal.

III. Status of Claims

The claims on appeal are claims 1-21, directed to methods and systems for using objects defined in a first object definition notation to access object definition information specified in a second object definition notation without translating the object definition

information specified in the second notation into the first notation. The appealed claims are pending and were finally rejected (Final Office Action, dated February 4, 2000). Appellant filed its Notice of Appeal with appropriate extensions of time on August 4, 2000.

IV. Status of Amendments

No amendments were filed subsequent to final rejection.

V. Summary of Invention

The invention relates to communication of object interface definition information between heterogeneous object-oriented or object-based distributed object systems. When objects defined in one notation (such as C++) are used to manipulate (for example to invoke or instantiate) objects defined in another notation (such as Smalltalk), object definition information, including interface information, must be somehow acquired across definition notations.

One approach is to manually or automatically translate interface definitions in a first notation into a second notation. When the first and second notations are programming languages, the original and translated interface definitions may be used to create a "wrapper object" having two sets of interfaces - one set in the first programming language and a second in the second programming language, together with program logic that translates invocations on the first (wrapper) interface to the second (wrapped) interface.

Another approach is to provide a programming-language-independent object interface definition notation, with mappings from the programming-language-independent notation into various programming languages. Object interfaces are then defined in the programming-language-independent notation and translated via the mapping into interface definitions in programming languages for which mappings are available. Using

these translations, objects in one programming language may invoke objects in another programming language.

One such programming-language-independent object interface definition language is Common Object Request Broker Architecture (CORBA) Interface Definition Language (IDL). Another is the notation defined for Common Management Information Protocol (CMIP) objects in accordance with CCITT Recommendation X.722 Guidelines for the Definition of Managed Objects (GDMO).

As with programming language object interface definitions, interface definitions in one programming-language-independent object interface definition language such as CORBA IDL may be translated to another programming-language-independent object interface definition language such as GDMO. In addition, programming-language-independent object interface definition languages may be used in conjunction with wrapper objects having interfaces defined in languages with mappings from the programming-language-independent interface definition languages.

Translating object interface definitions from one notation to another poses problems arising from semantic and syntactic differences between the notations. For example, GDMO supports a richer variety of types than does CORBA IDL. Translation into CORBA IDL often leads to unmanageable CORBA IDL definitions, and very large programs. Also, information is often lost during translation, and the resulting CORBA IDL may not include all information necessary to perform desired operations.

The present invention obviates the need for interface translation by encapsulating foreign object definition information in encapsulator objects having native interfaces without translating foreign object definition information into native object definition notation. The foreign object definition information may then be discovered by interrogating the encapsulator objects using their native interfaces. Application, at p. 6, line 7 - p. 7, line 25. Because no translation from the foreign notation to the native notation is required to create the encapsulator objects, encapsulator object creation is

unhindered by semantic and syntactic differences between notations. Further, the encapsulator objects may have predefined native interfaces that do not depend upon the foreign interface definitions which they encapsulate. Application, at p. 7, lines 21-24. Thus, the present invention may be used for an off-the-shelf gateway that need not be customized to the objects which will be encapsulated. Application, at p. 10, lines 6-17.

In a preferred embodiment, encapsulator objects are created by a parser that parses foreign notation. Application, at p. 6, lines 18-19. For example, in one preferred embodiment, the parser accepts foreign object definition information expressed in foreign notation and instantiates one or more encapsulator objects for each rule (such as a production) in a grammar that corresponds to the syntactic structure defined in the foreign notation. Application, at p. 6, lines 19-25. The resulting collection of encapsulator objects reflects the syntactic structure of the foreign object definition information. Application, at p. 6, lines 25-27. The encapsulator objects expose native interfaces that may be interrogated to discover the foreign object definition information they encapsulate. Application, at p. 6, lines 27-29.

The parser may be included in an object factory (an object which instantiates and initializes other objects) which instantiates native objects corresponding to the nonterminals of the syntax defined by the foreign object definitions. Application, at p. 7, lines 10-14. The resulting native objects may be interrogated by other native objects through their native interfaces to discover the encapsulated foreign object definitions. Application, at p. 7, lines 14-17. No translation of the encapsulated foreign object information into native notation is required, avoiding the problems posed by the differences between native and foreign syntax and semantics. Application, at p. 7, lines 17-20.

VI. Issues

The following issues are presented for review:

OBVIOUSNESS

(1) Whether claims 1-21 directed to methods and systems for using objects defined in a first object definition notation to access object definition information specified in a second object definition notation without translating the object definition information specified in the second notation into the first notation are obvious under 35 U.S.C. § 103(a) over Foody (U.S. Pat. No. 5,732,270) in view of Mowbray et al. ("The Essential CORBA: System Integration Using Distributed Objects," pp. 231-267). Because the Examiner conceded that Foody does not teach "without translating" limitation (February 4, 2000 Final Office Action at 2), it is sufficient to show that the use of object definition information specified in a foreign notation without translating the object definition information into a native notation is not disclosed or suggested by Mowbray.

(2) Whether claims 7-10, which are directed to methods for accessing object definition information stored in one or more software objects residing in computer memory, comprising the steps of invoking said one or more objects by means of at least one interface specified in a first notation, said one or more object returning in response to said invocation object definition information specified in a second notation, are obvious under 35 U.S.C. § 103(a) over Foody in view of Mowbray. Neither Foody nor Mowbray, alone or in combination disclose or suggest a method for accessing object definition information stored in one or more software objects residing in computer memory, comprising the steps of invoking said one or more objects by means of at least one interface specified in a first notation, said one or more object returning in response to said invocation object definition information specified in a second notation.

(3) Whether claim 21, which is directed to a method of constructing an object invocation comprising the steps of: instantiating an object collection of objects

corresponding to rules specifying the syntax of said object invocation; receiving information of the content of the object invocation; and interrogating the object collection with the information to determine a set of objects sufficient to construct the invocation, is obvious under 35 U.S.C. § 103(a) over Foody in view of Mowbray. Neither Foody nor Mowbray, alone or in combination disclose or suggest a method of constructing an object invocation comprising the steps of instantiating an object collection of objects corresponding to rules specifying the syntax of said object invocation; receiving information of the content of the object invocation; and interrogating the object collection with the information to determine a set of objects sufficient to construct the invocation.

VII. Grouping Of The Claims

A.

Claims 1-21 stand rejected under 35 U.S.C. § 103(a) as allegedly obvious over over Foody (U.S. Pat. No. 5,732,270) in view of Mowbray et al. ("The Essential CORBA: System Integration Using Distributed Objects," pp. 231-267). Appellant groups the claims as follows and respectfully submits that:

(1) Claims 1-21 are not obvious because neither Foody nor Mowbray, taken singly or in combination, disclose or suggest the use of object definition information specified in a foreign notation without translating the object definition information into a native notation. This "without translating" limitation forms a part of each of claims 1-21.

However, claims 1-21 do not stand or fall together with respect to the rejection on Foody in view of Mowbray, for the following reasons:

(2) Claims 7-10 are not obvious for the independent reason that the asserted combination fails to disclose or suggest a method for accessing object definition information stored in one or more software objects residing in computer memory, comprising the steps of invoking said one or more objects by means of at least one interface specified in a first notation, said one or more objects returning in response to said invocation object definition information specified in a second notation. This limitation forms a part of each of claims 7-10.

(3) Claim 21 is not obvious for the independent reason that the asserted combination fails to disclose or suggest a method of constructing an object invocation

comprising the steps of instantiating an object collection of objects corresponding to rules specifying the syntax of said object invocation; receiving information of the content of the object invocation; and interrogating the object collection with the information to determine a set of objects sufficient to construct the invocation. These limitations form a part of claim 21.

VIII. ARGUMENT

A. The Rejected Claims Are Not Obvious Over Foody Combined With Mowbray

Claims 1-21 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Foody (U.S. Pat. No. 5,732,270) in view of Mowbray et al. ("The Essential CORBA: System Integration Using Distributed Objects," pp. 231-267).

As set forth in detail below, the Examiner has failed to make out a *prima facie* case of obviousness over Foody combined with Mowbray.

1. The Teachings Of Foody

Foody discloses a system that enables objects from two or more heterogeneous object systems to interoperate. A native proxy object, indistinguishable by the native object system from other native objects, is constructed for the real foreign object. The proxy object contains an identifier of the real object, as well as a pointer to a software description of how to access and manipulate the object--e.g. how to call its methods, set its properties, and handle exceptions. When the proxy object is manipulated, it follows the instructions in the software description which, in turn, results in the corresponding manipulation of the foreign object. Foody, col. 6, line 47 - col. 7, line 2.

Foody uses an "Object Exporting Framework" to export object definitions. *See* Foody, col. 16, line 9 - col. 18, line 6. Foody describes only one way to export object definitions: "generating an interface definition in a 'language' appropriate to the foreign object system (such as Interface Definition Language or IDL in CORBA systems. . . .)"

Foody, col. 16, line 66 - col. 17, line 2. This is a method of translating an object definition from one object definition notation into another object definition notation.

In the February 4, 2000 Final Office Action, at 2, the Examiner conceded that "Foody does not teach without translating the object definition from the second notation into the first notation since Foody may involve generating an interface description in a language appropriate to the foreign object system." Since this limitation appears in all of the pending claims, Foody does not teach the claimed invention.

2. **The Teachings Of Mowbray, et al.**

The Mowbray reference, entitled "The Essential CORBA: Systems Integration Using Distributed Objects" is generally directed to the use of CORBA to integrate legacy systems. The portions of the Mowbray reference cited by the Examiner are directed to "object wrapping." Mowbray, at 231-267.

Mowbray states that

> Object wrapping allows us to provide access to a legacy system through an encapsulation layer. ***The encapsulation exposes only those attributes and operations definitions desired by the software architect.*** . . . The wrapper serves as an interoperability bridge between a legacy system and software architecture. On one side of the bridge, the wrapper communicates using the legacy system's existing communications facilities. On the other side of the bridge, the wrapper presents other applications a clean interface that provides abstract services. . . . The wrapper functions as a ***customized*** gateway between the legacy system and the software architecture.

Mowbray, at 232 (emphasis added). Mowbray explains further that "[o]bject wrapping is a practice that transforms a component's software interfaces from one form to another." Mowbray, at 238. Object wrapping is thus a *customized* translation of the interfaces of a legacy system to a new software architecture, performed by a software architect.

The examiner rejected claims 1-21 over Foody in view of Mowbray, stating that

> Mowbray teaches integrating object definitions of different notations (wrapping to provide layering of API's to legacy systems). Mowbray teaches two alternative encapsulating/wrapping techniques, one with substantial changes/translations between the notations, and one without modifying/translating the underlying

interfaces/notation. See page 232, last para. - page 233, 2nd para.; page 237, 2nd - 3rd paras; page 238, table.

February 4, 2000 Final Office Action, at 2. The language cited by the examiner describes two types of object wrapping: "Layering" and "Wrappers for Architecture Implementation." Mowbray, at 232, 237.

Both layering and wrappers for architectural implementation, and indeed all object wrappers, require the software architect to develop a customized mapping from the interfaces of the wrapped legacy system onto the interfaces of the wrapper object or objects. Layering is a simple mapping from one form of application program interface (API) to another, perhaps with substantial changes, such as mapping one set of operations to a completely different set. *See* Mowbray, at 233.

A Wrapper for architecture implementation is more than a simple encapsulation of an API, and requires translation and reimplementation in the wrapper system of an entire legacy software architecture. It must implement the wrapped legacy architecture design "in all aspects." Mowbray, at 236. A wrapper for architecture implementation is thus a more complete translation of the definitions of the legacy system than simple layering. Clients can use the wrapped architectural framework to communicate with legacy implementations that are compliant with the wrapped framework. Mowbray, at 237. In one example, a software architectural framework called DISCUS is wrapped by CORBA objects by mapping its functions onto OMG IDL. Mowbray, at 232, 237.

The object wrapper approach described in Mowbray is substantially different than the invention disclosed and claimed in the present application. Unlike object wrapping, the system of the present invention permits object definition information of any object defined in a foreign object notation to be discovered using *predefined* native interfaces that do not depend on the foreign object definitions. No customization or choices of attributes and operations to transform are required by a software architect. As a result, unlike object wrapping, which requires the software architect to translate the foreign

interfaces into native notation, an off-the-shelf implementation of the present invention can instantiate a collection of objects having predefined native interfaces that encapsulate the interface definitions of the single API or the entire legacy architecture, automatically and without translation.

3. The Rejection of Claims 1-21 over Foody in View of Mowbray, et al. Should Be Reversed Because the Combination Does Not Disclose or Suggest the "without translation" limitation

The rejection of claims 1-21 over Foody in view of Mowbray under 35 U.S.C. 103(a) is founded on a misconstruction of the word "translate" and its cognates as used in the claims, and on a misapplication of that limitation to Mowbray.

Mowbray explains that "[o]bject wrapping is a practice that transforms a component's software interfaces from one form to another." Mowbray, at 238. When used to transform an interface defined in one notation into an interface defined in another, object wrapping comprises "translating" within the meaning of the "without translating" limitation found in claims 1-21. Object wrapping is described at length in Mowbray, because the process of performing a customized mapping from a single legacy interface or an entire legacy architecture is complex and labor-intensive. The invention disclosed in the present application and claimed in claims 1-21 makes it possible to provide foreign object definition information via pre-defined native interfaces without translation, avoiding the need for the labor-intensive mapping required by object wrapping.

As pointed out by the examiner in an Advisory Action mailed August 14, 2000, the Microsoft Press Computer Dictionary defines "translate" to mean "1. In programming, to convert a program from one language to another." Microsoft Press Computer Dictionary, at 475 (3d. Ed., 1997). Clearly "a practice that transforms a component's software interfaces from one form to another" is translation if the transformation converts an interface from one language or notation to another. The present invention does not require such translation. Mowbray's object wrapping does.

Both layering and wrappers for architectural implementation involve mapping the interfaces of the wrapped legacy system onto the interfaces of the object wrapper system. Layering is repeatedly described as a mapping from one form of application programming interface (API) to another. *See* Mowbray at 233 ("A layer is a mapping from one form of application program interface (API) to another."), 237 ("A simple layer mapping from the legacy APIs to OMG IDL provides for a quick and powerful wrapping...."). And wrappers for architectural implementation must implement the legacy architecture design "in all aspects." (Mowbray, at 236.) For example, when referring to an example wrapper for architectural implementation (the DISCUS framework), Mowbray states "the wrapping of a legacy system may mean the mapping of some of its functions onto the framework Object Management Group Interface Definition Language operations." Mowbray, at 232.

To "map" is defined by the Microsoft Press Computer Dictionary as "[t]o *translate* one value into another." Microsoft Press Computer Dictionary, at 298 (3rd Ed., 1997) (emphasis added). Thus, to map the interfaces of the wrapped system onto the interfaces of the wrapper object system is to translate the wrapped system's interfaces from its native notation to the notation of the wrapper object.

In the advisory action mailed August 14, 2000, the Examiner stated that

> "[a]s to the argument that Mowbray's wrapping requires transformation (p.2-3), transformation changes the appearance or format of data, whereas translation converts from one language to another using a compiler/interpreter. Such distinction is well known to one of ordinary skill in the art and may be found in a computer dictionary (e.g., by MS Press). In other words, Mowbray uses wrapping/transforming, without translating. If applicant's translating means differently, it should be brought out in the claims.

The distinction between "transformation" and "translation" drawn by the examiner does not support the rejection. "Transformation" and "translation" are the same thing, if the "transformation" is from one notation or language to another.

As noted above, the Microsoft Press Computer Dictionary defines "translate" to mean "1. In programming, to convert a program from one language to another." Microsoft Press Computer Dictionary, at 475 (3d. Ed., 1997). It also defines "transform" to mean "1. To change the appearance or format of data without altering its content; that is, to encode information according to predefined rules." *Id.* To "change the appearance or format of "a program from one language or notation to another "without altering its content," (*See* definition of "transform") is "to convert a program from one language or notation to another" (*See* definition of "translate"). Therefore, to "transform" a program from one language or notation to another is to "translate" it from one language to another, as defined by the Microsoft Press Computer Dictionary. Mowbray's object wrapping thus requires translation as that term is used by those of ordinary skill in the art, as reflected in the Microsoft Press Computer Dictionary.

4. The Rejection of Claims 7-10 Over Foody And Mowbray, et al. should be reversed because the combination does not disclose or suggest the limitation of returning object definition information specified in a second notation

Claim 7 - 10 of the present application require the steps of invoking one or more objects by means of at least one interface specified in a first notation, said one or more object returning in response to said invocation object definition information specified in a second notation. Neither Foody nor Mowbray disclose or suggest this limitation. Rather, both references explicitly teach away from this limitation. In both Foody and Mowbray, the explicit goal is to make foreign objects indistinguishable from native objects by ***hiding*** foreign object definition information from native objects and making only translated object definition information available to native objects. Claims 7-10 are therefore not obvious over Foody in view of Mowbray for this additional reason.

Foody states that objects in a foreign object system "appear to be native to the object system in which they are used or accessed." Foody, col. 6, ll.51-53. These

objects are "indistinguishable from other native objects." Foody, col. 6, l.62. Similarly, Mowbray states that "[o]ne of the key benefits of wrapping is that services may be provided by a legacy system or by a new object. The clients cannot distinguish between the implementations." Mowbray, at 232. This is inconsistent with invoking one or more objects by means of at least one interface specified in a first notation, said one or more object returning in response to said invocation object definition information specified in a second notation, as required by claims 7-10.

5. The Rejection of Claim 21 Over Foody And Mowbray, et al. should be reversed because the combination does not disclose or suggest constructing an object invocation by instantiating a collection of objects specifying the syntax of the invocation and interrogating the collection of objects to determine a set of objects sufficient to construct the invocation

Claim 21 recites a method of constructing an object invocation comprising the steps of instantiating an object collection of objects corresponding to rules specifying the syntax of said object invocation; receiving information of the content of the object invocation; and interrogating the object collection with the information to determine a set of objects sufficient to construct the invocation. Neither Foody nor Mowbray disclose such a method, and in fact both references teach away from such a system.

Because the systems disclosed in Foody and Mowbray map the interfaces of foreign objects onto the interfaces of native objects, it is unnecessary to interrogate objects to determine a set of objects sufficient to construct an invocation. Rather, foreign objects are invoked by simply invoking proxy or wrapper objects, which are indistinguishable from other native objects. *See* Foody, col. 6, ll.51-53; col. 6, l.62; Mowbray, at 232. There is thus no motivation in either reference to provide the features of claim 21.

IX. CONCLUSION

Neither Mowbray nor Foody disclose or suggest the present invention, which provides foreign object definition information to a native object system without translating the foreign object information into a native notation. The claimed "without translation" feature of the present invention provides substantial benefits to users, avoiding entirely the laborious and complex mapping process required by Mowbray's object wrapper. Because the examiner misconstrued the meaning of the word "translate" and its cognates as used in the claims, and thereby misapplied the Mowbray reference, the rejection of Claims 1-21 over Foody in view of Mowbray should be reversed.

Mowbray and Foody disclose object systems in which foreign objects are invoked via wrapper or proxy objects which are otherwise indistinguishable from other native objects. The systems of Mowbray and Foody therefore do not render obvious claims 7-10 for the independent reason that they do not disclose or suggest a method comprising the steps of invoking one or more objects by means of at least one interface specified in a first notation, said one or more object returning in response to said invocation object definition information specified in a second notation. Similarly, Mowbray and Foody do not render claim 21 obvious for the independent reason that they do not disclose or suggest a method of constructing an object invocation comprising the steps of instantiating an object collection of objects corresponding to rules specifying the syntax of said object invocation; receiving information of the content of the object invocation; and interrogating the object collection with the information to determine a set of objects sufficient to construct the invocation.

All the rejections should be reversed.

Respectfully submitted,

Date: May 24, 2001

37,242

Garland T. Stephens (Reg. No.)

PENNIE & EDMONDS LLP
1155 Avenue of the Americas
New York, New York 10036-2711
(212) 790-9090